2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing

An Asynchronous Message Delivery Service for iGaDs (intelligent Guards against Disasters)

Y. Z. Ou, C. M. Huang, C. T. Hu, J. W. S. Liu
Institute of Information Science
Academia Sinica
Taipei, Taiwan
{yzou, charles.huang, tomhu, janeliu}@iis.sinica.edu.tw

E. T. H. Chu
Dept. of Computer Science
National Yunlin Univ. of Sci. and Tech.
Yunlin, Taiwan
edwardchu@yuntech.edu.tw

C. S. Shih
Dept. of CS. and Info. Eng.
National Taiwan Univ.
Taipei, Taiwan
cshih@csie.ntu.edu.tw

***Abstract*—This paper describes architecture and middleware for a system of intelligent Guards against Disasters, called iGaDs for short. iGaDs are smart devices and applications that can receive, authenticate and process standard-conforming disaster alert messages from authorized senders and respond by taking appropriate actions to help us to be better prepared for nature disasters. They are designed to be used ubiquitously as elements of future disaster-prepared smart homes and environments. The prototype prioritized asynchronous alert message delivery service described here is built by using a data bridge to connect Qpid and PubSubHubbub as a way to push alert messages to a large system of iGaDs via Internet.**

Keywords—intelligent things; message delivery service; smart environment; disaster preparedness

I. INTRODUCTION

Thanks to world-wide efforts in recent years on development and deployments of technologies for the predication and detection of nature disasters and ICT (Information and Communication Technology) infrastructures for generation and distribution of disaster alerts/warnings, most developed regions are now better prepared for disasters than even a few short years ago. Today, advanced weather radars and warning decision support systems (e.g., [1, 2]) enable accurate predictions of paths and severities of tornados tens of minutes in advance. In developed countries frequented by strong earthquakes (e.g., Taiwan, Japan and parts of USA and Mexico), broadband arrays of seismometers and strong seismic motion sensors [3, 4] are networked with computers running advanced auto-location and focal mechanism determination tools (e.g., [5-7]). Such networks of things can deliver early warnings of strong earthquakes within seconds after their occurrences, providing people and things with warnings a second or more before shock waves arrive and ground motion starts. We have witnessed time and again the working of tsunamis detection and predication technology, serving people near strong earthquakes, as well as people an ocean away.

We now also have standards and tools developed by large integrated projects and standards organizations (e.g., [8-13]) to support interoperability of diverse sensor networks and sensor webs. For more than a year now, Integrated Public Alert and Warning System – Open Platform for Emergency Networks (called IPAWS-OPEN [14] for short), has been providing services in the USA to receive and authenticate standard-based messages from alerting authorities and then broadcasts the messages via all communication pathways, including digital radio broadcast, cellular networks and Internet to emergency alert systems. Public warning systems, such as Global Disaster Alert and Coordination System (GDACS) [15], European Public Warning System (EU-Alert) [16] and others [17, 18], have been used or are being developed to provide similar emergency alert services elsewhere in the world. XML-based EDXL (Emergency Data Exchange Language) [19] messaging standards, including CAP (Common Alerting Protocol) [20] EDXL-DE (Distribution Element), and EDXL-RM (Resource Messaging), enable information exchanges between emergency information systems and public safety organizations, automatic report by sensor systems to analysis centers, and aggregation and correlation of warnings from multiple sources. Consequently, alert decision support systems not only can generate more accurate alerts earlier but also can have the alerts delivered and understood sooner.

In comparison, technologies to take advantage of machine-readable and authenticable alert messages remain immature. Smart and intelligent homes and environments now offer us devices, applications and services for our comfort, convenience, and social connectivity, but nothing to help us prevent loss of lives, reduce chance of injuries and minimize property damages and economical losses when disasters strike. This fact motivated us to propose the pervasive use of *iGaDs* (*intelligent Guards against Disasters*) as elements of future disaster prepared smart home and environment [21]. The term iGaDs refers to embedded devices, systems, and applications that can authenticate and process standard-conforming disaster warning messages and respond by taking appropriate actions to enhance our preparedness for disasters. Hereafter, for sake of concreteness, we assume in subsequent discussions that alert messages conform to the CAP standard and sometimes call

978-0-7695-4974-3/13 $26.00 © 2013 IEEE
DOI 10.1109/IMIS.2013.152

iGaDs *CAP-aware* devices. We will use the terms messages, alerts and data interchangeable when there is no confusion.

As examples, Fig. 1 shows two embedded iGaDs used in buildings and homes: They are designed to respond to warnings of imminent strong earthquakes. When alerted by a CAP message warning of an earthquake of a specified magnitude or stronger, a CAP-aware elevator controller in a smart building slows down elevators, stops them and opens their doors when they reach the closest floors. An iGaD in a smart home shuts down natural gas intake valve to prevent fire and open the front door to ease evacuation. Such an iGaD can be configured to respond to strong tornado warnings as well as earthquake warnings. We will return to elaborate this point later. In addition, software iGaDs are CAP-aware applications that run on computers, smart phones and other platforms with sufficient computing power and memory space and some form of location-based service. Examples can be found in [21].

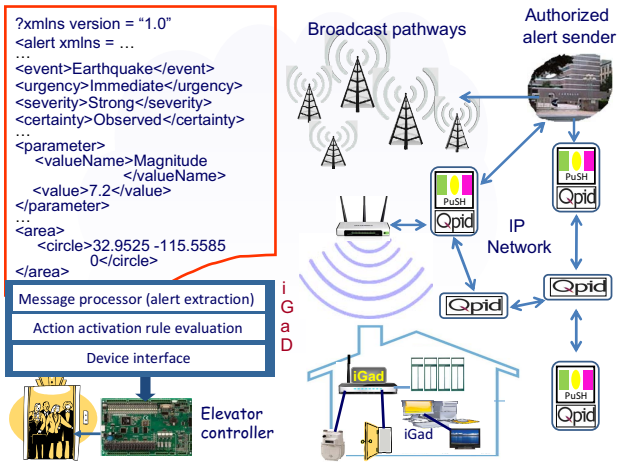

Fig. 1. Examples of embedded iGaDs

Our previous work on iGaDs [21, 22] focused primarily on architecture and key components of diverse iGaDs and means to make them configurable and customizable at installation and maintenance times. We assumed there that messages alerting of all types of disasters are delivered to CAP-aware devices and applications via cellular cast and broadcast and hence the hardware enhancements discussed in [22] are needed to keep energy consumption of battery operated iGaDs low.

This paper focuses on the case of iGaDs linked by Internet. The prototype asynchronous message delivery service described here is built by using a data bridge middleware to link Qpid [23, 24] with PubSubHubbub [25]. The resultant platform enables alert messages to be pushed asynchronously in real-time via Internet to numerous and diverse iGaDs. In addition to asynchronous message delivery, the problems addressed by the service and the middleware are how to improve the chance of timely delivery of more urgent alert messages (e.g., alerts about earthquakes and tornados with required latency from a fraction of a second to minutes) at the expense of less urgent ones (e.g., about wide-area floods that take days to develop).

Following this introduction, Section II describes related works. To make this paper sufficiently self-contained, Section III presents an overview of reference architecture and components of configurable and customizable iGaDs. Section IV presents the structure, design and implementation of prototype message delivery service. Section V summarizes the paper and discusses future works.

II. Related Works

As Fig. 1 shows, each iGaD has a message processor that parses alert messages in a standard format to extract information about the alerts contained in the messages. This part closely resembles CAP-EAS decoders in public emergency alerting systems (EAS) [26] used in the USA. CAP being XML-based, iGaD message processor is essentially an XML parser. Software iGaDs have a wide range of choices among matured XML parsers for operating systems and programming languages supported by popular computer and mobile platforms (e.g., [27-30]). Existing hardware parsers (e.g., [31]) are typically designed for applications that need high throughput and processing speed. In contrast, we need lightweight parsers that can extract alert information from CAP messages within a fraction of a second with minimum energy. We plan to develop such a parser in the future.

Figure 1 also shows that each iGaD needs to evaluate rules governing whether it should respond to the alert extracted from the current message and if it should respond what action to take. Rather than using existing rule engines (e.g., [31-34]) for this purpose, the current version of iGaD prototype uses a lightweight scheme to processing the rules [22]. iGaDs based on this scheme can be easily customized.

As stated earlier, our message delivery service for pushing alerts asynchronously in real-time over Internet to embedded and mobile iGaDs is built by linking Qpid [23] and PubSubHubbub (PuSH) [25]. Qpid implements Advanced Message Queuing Protocol (AMQP) [24], which can deliver messages by Publish and Subscribe message protocol. Qpid also provides transaction management, prioritized message queuing, distribution, security and many other features. PubSubHubbub is a simple and flexible server-to-server, web-hook-based publish and subscribe protocol. Servers speaking the PuSH protocol can get near instant notifications (via web-hook callbacks) when a *topic* (i.e., a feed URL) subscribed by them is updated. However, PuSH by itself is not ideal for delivering alerts to a large number of iGaDs for many reasons, including the ones discussed in [35]. By integrating Push with Qpid, we aim to make the platform more efficient, resilient and responsive, especially when the network is damaged during disasters. Section IV will describe the prototype.

III. iGaDs Architecture and Design Rationales

As discussed in [21, 22], iGaDs of the same type may be required to respond to the same alert differently, depending on

where and how they are used. For this reason, they must be highly configurable and customizable.

A. Action Activation Rules and Configuration Parameters

To illustrate, let consider two iGaDs of the same type. One is used to guard an entry-access controlled door of an office building that is a designated public severe storm shelter. The door is normally locked. When alerted of an EF (Enhanced Fujita) 4 or 5 tornado, the iGaD is required to respond by unlocking the door in order to make the building accessible to all people. The iGaD also unlocks the door in response to alerts of earthquakes magnitude 8 or stronger to ease the evacuation of people in the building. In other words, the iGaD shall send an unlock command to the building door in response to an alert message if either one or both of the following rules is true. Such rules are called *action activation rules* of the iGaD.

(1) `(EventType == "Earthquake") AND (Scale >= THRESHOLD_MAGNITUDE)`

(2) `(EventType == "Tornado") AND (Scale >= THRESHOLD_SEVERITY)`

The values of variables EventType and Scale in the expressions of these rules are the standard name of the event (e.g., earthquake, tornado, tropical cyclone, etc.) and the value of the severity measure (e.g., in the USA, Modified Mercalli (MM) intensity scale for earthquakes, Enhanced Fujita (EF) scale for strength of tornados, and category for hurricanes, etc.) specified by the alert. They are extracted from the message before the rules are evaluated. Parameters with all capital-letter names are configuration parameters of the iGaD. Here, THRESHOLD_MAGNITUDE (e.g., 8 MM) is the maximum earthquake magnitude the building is constructed to withstand, and THRESHOLD_SEVERITY (e.g., EF 4) is the minimum severity of tornados for which the building is used as a public shelter. By setting these parameters at installation time and maintenance time, we can customize the iGaD according to the construction, condition and usage of the building.

In addition to changing configuration parameters that define the rules, action decisions of individual iGaDs of the same type can also be customized by providing them with individualized rules. To illustrate, we suppose that the second iGaD of the same type (as the iGaD for shelter door) is used for outside doors of a smart home. This iGaD should also command the doors to open in response to earthquakes stronger than the threshold scale for the home. During a tornado emergency, however, the outside doors should open in order to equalize air pressures in and out of the house, but only if and when a tornado actually hits the house. We can take into account of this consideration by letting the iGaD have the additional rule defined in terms of the readings of digital barometers inside and outside of the house. Rule (3) stated below is an example: The iGaD sends open commands to the outside doors when rule (1) is true or when rules (2) and (3) are both true.

(3) `InsideAirPressure >= OutsideAirPressure * THRESHOLD_RATIO`

B. Major Components

These examples motivated us to use a light-weight rule engine to provide all iGaDs with decision support. Figure 2 shows how the rule engine and the CAP message processor fit together with other components in a general structure of all embedded iGaDs. Solid arrows represent information flow among the components. A software iGaD does not have a device controller; a CAP-aware application takes its place.

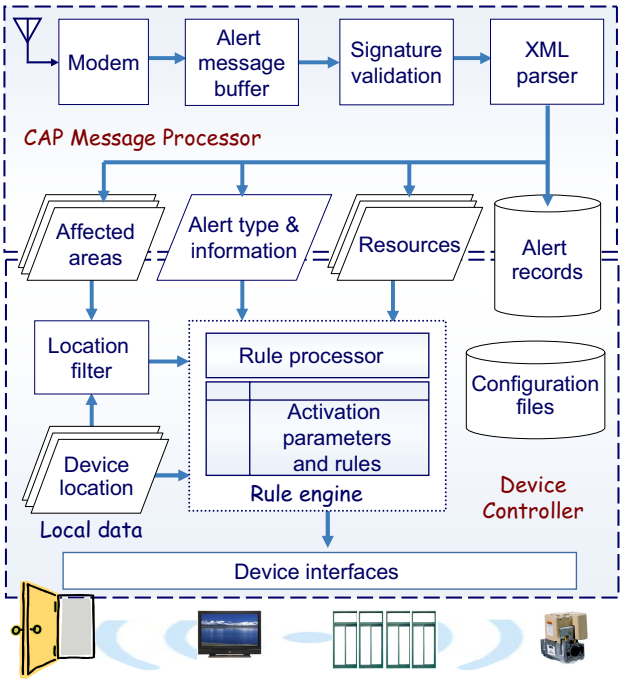

Fig. 2. General structure and major components

The CAP message processor is essentially the same for all iGaDs. It is responsible for extracting from each alert message the information needed by the iGaD device controller (or application) to decide whether and how to respond: The information extracted by the XML parser includes the name and scale of the alert event type and severity of the event, as well as specifications of areas affected by the alert. The message may also provide resources such as human-readable descriptions and URLs of files containing supplement information (e.g., photos, maps, audio, and so on) that may be useful to the public EAS and some iGaDs.

In addition to making decisions regarding actions taken by the iGaD, the device controller controls one or more physical device(s) (e.g., automatic lock and gas valve) connected physically to it, or via one or more networks. The device controller interacts with the driver of each physical device via events: The device driver handles events from the iGaD as commands (e.g., lock and unlock commands), and events from the drivers are treated by the iGaD as acknowledgements. Some iGaDs also relies on local data (e.g., device location, local sensor data, etc.) provided by its operating environment

for this purpose, as shown by Fig. 2. The figure omits interfaces to them as well.

An alert message may include additional parameters of the event, such as expiration times, image and text descriptions, and so on. Some iGaDs (e.g., those that provide emergency alert functions of building management systems) can use them as suggested by the CAP-EAS implementation guidelines [26] in the generation of their responses.

IV. Message Delivery Service and Platform

As stated earlier, the message delivery service described in this paper is designed to enable asynchronous deliveries of alert messages over Internet to iGaDs. This section describes its major components, interactions between the components, and prioritized capability of message delivery.

A. Hub, Data Bridge and Qpid

Figure 3 shows the major components in each node on the edge of a network that provides our message delivery service: We have PubSubHubbub (PuSH) hub on the top, Qpid at the bottom. They are linked by the data bridge middleware shown the middle of the figure. The data bridge combines PuSH and Qpid protocols so that the applications can take advantage of the application interface provided by PuSH and the prioritized asynchronous message delivery provided by Qpid. The right part of Fig. 1 shows how nodes containing this combination of components work together in a Pub/Sub network. (Three such nodes are shown in Fig. 1.) On the edge, nodes with PuSH services are deployed to provide web-based services to publishers, including authorized alert senders, and subscribers, including public EAS and aggregation servers for embedded devices. In the core of the IP network, only Qpid is deployed to route messages. (Figure 1 shows two such nodes.)

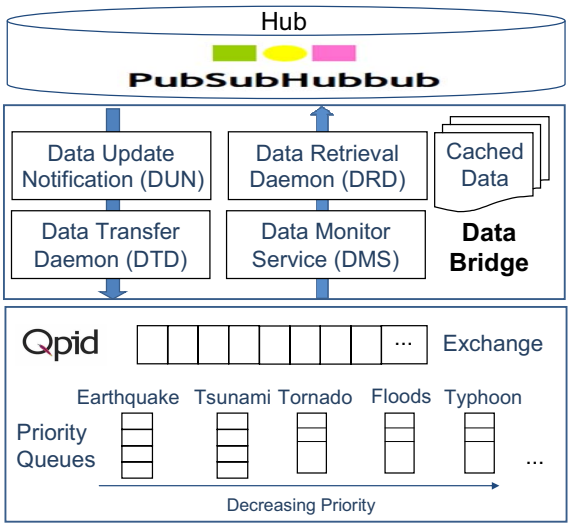

Fig. 3. Structure of Hub, Data Bridge and Qpid

Specifically, PuSH provides a HTTP-based interface using which publishers can publish data via topics defined by feed URLs and subscribers subscribe to selected topics: When a publisher wants to publish the latest of its data via a PuSH hub, it sends to the hub a HTTP POST request along with a callback URL to inform the hub that it has new data to publish. When thus notified, the hub retrieves the data from the publisher. We will return shortly to describe the actions taken by the hub and its interactions with the data bridge after it retrieved the data.

As we can see from Fig. 3, each Qpid has a set of data queues. Each data queue is dedicated to a topic. For our application, we may have data queues dedicated to topics on different types of disaster event. As examples, the figure shows data queues dedicated to topics on earthquake, tsunami, tornado, and so on. Qpid supports priority queues, using which messages are delivered in order of their priorities. We are experimenting with a scheduling scheme which assigns priorities of alert messages/data on the basis of the disaster types within the alerts. With this scheme, the relative priorities of data queues may be as shown in Fig. 3. The Qpid component also has an exchange, which routes messages from applications (in our case the data bridge) to data queues. The exchange is of topic type. Being a topic exchange, it gives us the flexibility of routing each alert message to one or more queues based on matching between a message routing key (e.g., disaster type) and other patterns.

The middle part of Fig. 3 depicts the structure of the data bridge. The rectangular boxes and arrows in the figure represent the functional modules and the direction of data flow, respectively. The modules named Data Update Notification (DUN) and Data Transfer Daemon (DTD) are used to move data from the hub served by the data bridge to the Qpid connected to the bridge: DUN is called by the hub when new data is available in a hub. When called, the module caches the latest data from the calling hub and saves it in the middleware. Then, DTD reads the data from the new file and move the data into the Qpid exchange by calling the native Qpid API. Here, we are able to keep the modification to the original design of PuSH hub at a minimum. (In fact, we only add a line of code of the hub to call DUN when new data is available.)

The second part of the data bridge is composed of Data Retrieval Daemon (DRD) and Data Monitor Service (DMS). The components are used to handle incoming data to the Qpid. Using the long polling mechanism supported by Qpid, DMS creates one or more threads to watch each data queue. As an example, suppose that an earthquake alert is announced and the alert message is forwarded to the Qpid and placed in the Earthquake data queue. DMS is informed by the return data of long polling. It activates DRD, which oversees the retrieve the message from the Earthquake data queue and sends the message and topic to the Hub. Then, hub finds out the subscribers of this topic and sends the message to the subscribers through a HTTP POST request.

B. *Interactions between PuSH, Data Bridge and Qpid*

The sequence diagram in Fig. 4 illustrates the interactions between a publisher, hub, the data bridge, Qpid and subscribers. Vertical arrows in the diagram indicate the direction of time flow, while horizontal arrows represent actions taken by parties participating in the interactions. One or more horizontal arrows labeled ***A1***, ***A2*** and ***A3*** illustrate the series of actions triggered by a publisher with new data to public. Horizontal arrows labeled ***B1*** and ***B2*** represent actions taken by components of the node when new data is forwarded to the Qpid from other Qpids.

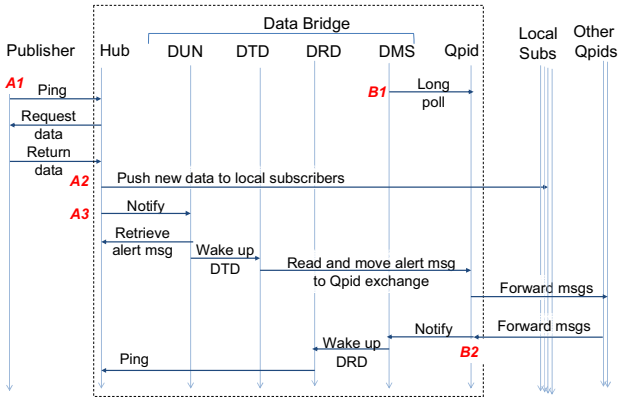

Fig. 4. Sequence Diagram of PuSH, Middleware and Qpid

The series of actions represented by the three horizontal arrows beginning from ***A1*** are according to the HTTP-based interface of the hub: The actions start from when a publisher pings the hub by sending to it a HTTP POST and a callback URL, indicating to the hub of new data on a topic being available. Thus notified, the hub actively retrieves the new data from the publisher according to the PuSH protocol.

Then the hub performs two actions. First, it pushes the new data to local subscribers (***A2***) of the topic. By local subscribers, we mean subscribers of the topic previously registered with the hub, and hence their URLs are maintained by the hub. Second, the hub notifies the data bridge. This action, labeled ***A3*** in Fig. 4, triggers the series of actions by DUN and DTD components of the data bridge to move the new data, referred to as a new alert message in the diagram, to the exchange and from there to data queue(s) on the Qpid. The hub notifies the data bridge by calling the notification API function of DUN with the updated topic (disaster type) as an input parameter. Upon notification, DUN retrieves the new message from the hub and saves it temporarily in the data bridge. It then wakes up the data transfer daemon (DTD) to do the actual data transfer work. Once the message is routed to a data queue in the Qpid, it is forwarded by Qpid to other Qpid(s) according to the routing table of the Qpid.

Horizontal arrows labeled ***B1*** and ***B2*** illustrates the actions of the components when new a new message reaches the local Qpid from other Qpids. As ***B***1 indicates, DMS uses long polling mechanism to monitor each data queue of the local Qpid for message arrivals from other Qpids. Upon awaken by the arrival of a new message to a Qpid data queue, the monitoring thread in turn wakes up the data retrieval daemon to retrieve incoming message from the data queue and ping the hub. In essential, the local data bridge acts as a publisher. Its notification to the hub triggers ***A1*** and subsequently ***A2.*** In this case, the hub does not do ***A3***.

C. *Prioritized Message Delivery*

We conclude this section by calling attention to the multiple data queues with different priorities for different topics depicted by Fig. 3. This queuing mechanism provided by Qpid allows us to schedule messages/data deliveries according to any fixed priority scheme. The priority scheme shown in the figure is an example. This capability is essential for our application. During a disaster, some links within the network may be unavailable or unstable, limiting network bandwidth to more or less degree, while some alert messages (e.g., alerts of imminent strong earthquake) must be delivered in time. Sending different types of alert and associated data with different quality of service requirements on priority basis is a try-and-true way to meet such requirements.

Relying solely on Qpid for timely message delivery is not sufficient, clearly: Data transfers between the hub and Qpid must also be prioritized. Due to space limitation, Fig. 3 omits the priority queues in the data bridge, a set of queues for each direction of data transfer. The DTD and DRD modules share a pool of worker threads and use them to carry out the data transfer work. When a worker thread finishes transferring a message to (or from) Qpid from (or to) the hub, it may choose to transfer the data waiting in the highest priority non-empty downward queue or in the highest priority non-empty upward queue. The choice depends on the scheduling scheme used by the data bridge to arbitrary resource contentions among them. The middleware is designed so that priority assignment (and hence scheduling choices) can be made at initialization time as well as dynamically when network connectivity and traffic condition change. We will experiment with and evaluate alternative end-to-end priority schemes in the future.

V. SUMMARY AND FUTURE WORK

The previous section described an asynchronous messaging service for real-time, prioritized deliveries of alert messages to iGaDs connected via Internet. The service uses a data bridge to connect a PuSH hub on the top to a Qpid at the bottom in each node on the edge of the network. In this way, the service enables authorized alert message senders to publish alerts via well known feed URLs and then leverages the prioritized asynchronous message delivery capability of the Qpid to make the messages available to message consumers who subscribed to the known feed URL.

The design and implementation of the message delivery service is a part of our effort to develop a proof-of-concept prototype system of diverse iGaDs that are linked to alerting authorities via broadcast networks and Internet. Our goal at this point in time is to demonstrate the feasibility of iGaDs and evaluate their effectiveness in representative scenarios. In this stage, we plan to experiment with current version of the

message delivery service when used in a disaster alert system where there are a relative small number of publishers, and the publisher uses the message delivery service via IPAWS. Rather than having individual iGaDs subscribe to their feed URLs, the subscribers are public emergency alert systems (EAS) [26] and commercial mobile alert services (CMAS). Each EAS or CMAS then disseminates the alert messages to iGaDs in its coverage area via broadcast.

As mentioned in the previous section, how responsive a network of nodes of built from PuSH, data bridge and Qpid can be depends, among other factors, how well system resources of each node are allocated to tasks that move data within the node. In the near future we plan to evaluate via simulation and measurements alternative resource allocation schemes to find the one(s) well suited for this application.

We also plan to investigate how to have the message delivery service serve iGaDs directly. This work involves the design of a large distributed system of Qpid nodes, each handling message deliveries to a subset of iGaDs subscribers. Challenging problems include load balancing and quality of service management.

ACKNOWLEDGMENT

This work is supported by the Academia Sinica project OpenISDM (Open Information Systems for Disaster Management)

REFERENCES

[1] "Case study on developing new technologies to increase tornado warning time," Georgia Tech, Severe Storms Research Center, http://www.gtri.gatech.edu/casestudy/twister-technology

[2] D. Coulter and T. Phillips, "New GOES-R to give more tornado warning time," http://ephemeris.sjaa.net/1108/d.html

[3] Seismometer, http://en.wikipedia.org/wiki/Seismometer

[4] T. C. Shin, et al., "Strong motion instrumentation programs in Taiwan," in Handbook of Earthquake and Engineering Seismology, W. H. K. Lee, H. Kanamori and P. C. Jennings, Ed. Academic Press, 2003 and BATS (Broadband Array in Taiwan for Seismology), http://bats.earth.sinica.edu.tw

[5] Focal mechanism, http://en.wikipedia.org/wiki/Focal_mechanism

[6] G. P. Hayes, L. Rivera, and H. Kanamori, "Source inversion of the W-Phase: real-time implementation and extension to low magnitudes," *Bull. Seism. Soc. Am.*, 80, 2009.

[7] N. C. Hsiao, et al., "Development of earthquake early warning system in Taiwan," *Geophys. Research Letters,* 36, 2009.

[8] C. Buratti, A. Conti, D. Darkari, and B. Verdone, "An overview on wireless sensor networks technology," Sensors, 2009.

[9] R. Sherwood and S. Chien, "Sensor Web: a new paradigm for operations," in Proceedings of International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations, June 2007

[10] SANY - an open service architecture for sensor networks, edited by M. Klopfer and I. Simons, http://sany-ip.eu/publications/3317, 2009.

[11] J. Strand, "SensorNet," presentation at http://www.ittc.ku.edu/workshops/sensornet/john_strand.pdf

[12] M. Botts, *et al*,, "OGC® sensor web enablement: Overview and high level architecture," http://www.opengeospatial.org/pressroom/papers

[13] OASIS (Organization for the Advancement of Structured Information Standards), http://www.oasis-open.org/

[14] IPAWS-OPEN (Integrated Public Alert and Warning System – Open Platform for Emergency Networks), at http://www.fema.gov/emergency/ipaws/about.shtm

[15] Global Disaster Alert and Coordination System, GDACS, http://w3.gdacs.org/

[16] European Public Warning System (EU-Alert) Using Cell Broadcast, ETSI TS 102900 V1.1.1 Technical Specification, ETSI, 2010.

[17] Worldwide PWS Initiative, http://www.one2many.eu/en/portfolio/emergency-alerts/worldwide-initiatives

[18] "Survey on Public Warning Systems in Europe," publication of European Emergency Number Association, December 2012.

[19] EDXL-DE: Emergency Data Exchange Language Distribution Element, V1.0, at http://www.oasis-open.org/committees/download.php/17227/EDXL-DE_Spec_v1.0.html

[20] CAP: Common Alerting Protocol, V1.2, http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html

[21] J. W. S. Liu, E. T.-H. Chu and C. S. Shih, "Cyber-physical elements of disaster prepared smart environments," IEEE *Computer,* vol.46, no.2, pp.69,75, Feb. 2013.

[22] W. P. Liao, Y. Z. Ou, E. T. H. Chu, C. S. Shih, and J. W. S. Liu, "Ubiquitous Smart Devices and Applications for Disaster Preparedness," *Proceedings of The 2012 International Symposium on UbiCom Frontiers - Innovative Research, Systems and Technologies,* September 2012.

[23] The Apache Software Foundation, "Apache QpidTM Open Source AMQP Messaging," http://qpid.apache.org/index.html , 11/27/2012.

[24] The Apache Software Foundation, "AMQP Messaging Broker (Implemented in C++)," http://qpid.apache.org/books/0.16/AMQP-Messaging-Broker-CPP-Book/pdf/AMQP-Messaging-Broker-CPP-Book.pdf , 11/27/2012.

[25] Bradfitz, Bslatkin, Andyster and Bradfitzgoog, "pubsubhubbub," https://code.google.com/p/pubsubhubbub/ , 11/27/2012.

[26] CAP-EAS Implementation Guideline, CAP-EAS Industry Group, 2010.

[27] NSXML Parser, at https://developer.apple.com/library/mac/documentation/cocoa/conceptual/nsxml_concepts/NSXML_Concepts.pdf

[28] libxml2, at http://xmlsoft.org/

[29] SAXParser, at http://docs.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/SAXParser.html

[30] LINQ to XML at http://msdn.microsoft.com/en-us/library/system.xml.linq.aspx

[31] E. C. Chee, E. Mohd-Yasin, A. K. Mustaph, "RBStrex: Hardware XML parser for embedded systems," International Conference for Internet Technology and Secure Transaction, 2009.

[32] JESS, the Rule Engine for the Java Platform , at http://herzberg.ca.sandia.gov/

[33] E. D. Schmidt, "Logician: A table-based rules engine suite in C++/.NET/Javascript using XML," at http://www.codeproject.com/Articles/194167/Logician-A-Table-based-Rules-Engine-Suite-In-C-NET

[34] C. Choi, et al., "MiRE: a minimal rule engine for context-aware mobile devices," Proceedings of the 3rd International Conference on Digital Information Management, November 2008.

[35] M. Tyler, "PubSubHubbub versus client push technologies, what is the difference, http://blog.caplin.com/2011/01/13/pubsubhubbub-vs-client-push-technologies-whats-the-difference/